Intermediate Java Programming Course

Course Overview
This course is designed for individuals who have a basic understanding of Java and want to deepen their knowledge and skills. The course covers intermediate concepts, including object-oriented programming, exception handling, collections, streams, and more. Each module includes code examples to illustrate the concepts discussed.

Course Structure
- **Module 1: Object-Oriented Programming (OOP) Concepts**
- **Module 2: Exception Handling**
- **Module 3: Collections Framework**
- **Module 4: Java Streams and Lambda Expressions**
- **Module 5: File I/O and Serialization**
- **Module 6: Multithreading and Concurrency**
- **Module 7: Java Networking**
- **Module 8: Java Database Connectivity (JDBC)**
- **Module 9: Java GUI Programming with Swing**
- **Module 10: Unit Testing with JUnit**

Module 1: Object-Oriented Programming (OOP) Concepts

Overview
In this module, we will explore the four main principles of OOP: Encapsulation, Inheritance, Polymorphism, and Abstraction.

Code Example
```java
// Encapsulation
class Employee {
    private String name;
    private int age;

    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}

// Inheritance
class Manager extends Employee {
    private String department;

    public Manager(String name, int age, String department) {
        super(name, age);
        this.department = department;
    }

    public String getDepartment() {
        return department;
    }
}

// Polymorphism
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
}

class Circle extends Shape {
    public void draw() {
        System.out.println("Drawing a circle");
    }
}

public class Main {
    public static void main(String[] args) {
        Shape shape = new Circle();
        shape.draw(); // Output: Drawing a circle
    }
}
```

Module 2: Exception Handling

Overview
This module covers how to handle exceptions in Java using try-catch blocks, finally clauses, and custom exceptions.

Code Example
```java
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            int result = divide(10, 0);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero: " + e.getMessage());
        } finally {
            System.out.println("Execution completed.");
        }
    }

    public static int divide(int a, int b) {
        return a / b;
    }
}
```

Module 3: Collections Framework

Overview
Learn about the Java Collections Framework, including lists, sets, maps, and their implementations.

Code Example
```java
import java.util.*;

public class CollectionsExample {
    public static void main(String[] args) {
        // List
        List<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        System.out.println("Names: " + names);

        // Set
        Set<String> uniqueNames = new HashSet<>(names);
        uniqueNames.add("Alice"); // Duplicate, won't be added
        System.out.println("Unique Names: " + uniqueNames);

        // Map
        Map<String, Integer> ageMap = new HashMap<>();
        ageMap.put("Alice", 30);
        ageMap.put("Bob", 25);
        System.out.println("Age of Alice: " + ageMap.get("Alice"));
    }
}
```

Module 4: Java Streams and Lambda Expressions

Overview
This module introduces Java Streams and Lambda expressions for functional programming.

Code Example
```java
import java.util.*;
import java.util.stream.*;

public class StreamsExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David");

        // Using Streams and Lambda
        List<String> filteredNames = names.stream()
                                          .filter(name -> name.startsWith("A"))
                                          .collect(Collectors.toList());

        System.out.println("Filtered Names: " + filteredNames);
    }
}
```

Module 5: File I/O and Serialization

Overview
Learn how to read from and write to files, as well as how to serialize and deserialize objects.

Code Example
```java
import java.io.*;

public class FileIOExample {
    public static void main(String[] args) {
        String filename = "example.txt";

        // Writing to a file
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
            writer.write("Hello, World!");
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Reading from a file
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Module 6: Multithreading and Concurrency

Overview
This module covers the basics of multithreading, thread lifecycle, and synchronization.

Code Example
```java
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread " + Thread.currentThread().getName() + " is running.");
    }
}

public class MultithreadingExample {
    public static void main(String[] args) {
        MyThread thread1 = new MyThread();
        MyThread thread2 = new MyThread();

        thread1.start();
        thread2.start();
    }
}
```

Module 7: Java Networking

Overview
Learn how to create simple client-server applications using Java's networking capabilities.

Code Example
```java
import java.io.*;
import java.net.*;

public class SimpleServer {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(12345)) {
            System.out.println("Server is listening on port 12345");
            Socket socket = serverSocket.accept();
            System.out.println("Client connected");
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            out.println("Hello from server!");

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Module 8: Java Database Connectivity (JDBC)

Overview
This module covers how to connect to a database using JDBC and perform CRUD operations.

Code Example
```java
import java.sql.*;

public class JDBCExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "root";
        String password = "password";

        try (Connection conn = DriverManager.getConnection(url, user, password)) {
            Statement stmt = conn.createStatement();
            ResultSet
```